In Hardware We Trust? From TPM to Enclave Computing on RISC-V

Emmanuel Stapf
Technical University of Darmstadt
Darmstadt, Germany
emmanuel.stapf@sanctuary.dev

Patrick Jauernig
Technical University of Darmstadt
Darmstadt, Germany
patrick.jauernig@sanctuary.dev

Ferdinand Brasser
Technical University of Darmstadt
Darmstadt, Germany
ferdinand.brasser@sanctuary.dev

Ahmad-Reza Sadeghi
Technical University of Darmstadt
Darmstadt, Germany
ahmad.sadeghi@trust.tu-darmstadt.de

***Abstract*—System-on-Chip platforms have been increasingly extended with trusted computing functionality to provide strong protection for sensitive software applications through *enclaves* that only require trust in the hardware and minimal software components. However, the deployed enclave architectures are still suffering from various shortcomings such as the lack of secure I/O, or being vulnerable to side-channel attacks. Thus, recent research works propose new enclave architectures with more comprehensive threat models and advanced security features. A majority of these solutions is being developed on the open RISC-V architecture. In this paper, we present a brief overview of the RISC-V-based enclave architectures, discuss their features, limitations and open challenges.**

***Index Terms*—RISC-V, Trusted Execution Environment (TEE), Enclave, Side-Channel Security**

I. Introduction

For decades we have been witnessing a never-ending arms race on modern software between (run-time) attacks and corresponding defenses. Defense approaches range from (fine-grained) address space layout randomization to control-flow integrity (CFI) or code-pointer integrity (CPI), and data-flow integrity. Each of these techniques has its own pros and cons, but all of them struggle to find the adequate security and efficiency trade-off. In addition, modern software is becoming more and more complex, and hence, the attack surface is growing steadily. In particular, large code bases such as operating systems (OS), have already been shown to be vulnerable to various attacks, and thus, are unsuitable to serve as an underlying Trusted Computing Base (TCB) for protecting sensitive services. In this context, Trusted Computing technology has promised significant improvement in protecting modern software by integrating hardware-assisted security primitives and components tightly into the System-on-Chip (SoC) and hardware platforms. This ranges from Trusted Platform Modules (TPMs) [29] to hardware instruction extensions for CFI [11], capability systems [14], [33] and Trusted Execution Environments (TEEs). TEE architectures commonly leverage hardware and software security primitives to enable isolated environments, usually called *enclaves*. Enclaves are used to isolate the execution of sensitive services from all other software, including the OS, and thus, protect them even against strong software adversaries. Only a small software TCB, which configures the underlying hardware security primitives of the system and manages these enclaves, is inherently trusted.

This work was funded by the German Federal Ministry of Education and Research under the funding number 16KIS1417. This publication is an author's copy.

Enclave-based security architectures have been proposed for a variety of computing platforms, ranging from resource-constrained microcontrollers and embedded systems, such as Sancus [24], TyTAN [4], TrustLite [19], Sanctum [10], Keystone [20] TIMBER-V [31], or CURE [2], to high-performance computing systems, e.g., industry solutions like Intel SGX [16], AMD SEV [17], ARM TrustZone [1], or academic solutions such as Sanctuary [5]. While industry solutions successfully enable a new level of protection against a more privileged software adversary, they often lack important features, such as secure I/O or protection mechanisms against sophisticated software attacks, e.g., cache side-channel attacks, which are typically not included in their threat models.

The advent of open-source hardware and the open architecture RISC-V initiated a new line of research and an opportunity to explore, scrutinize and design enclave architectures across the full stack, both hardware and software. This brought rise to a number of academic solutions such as Sanctum [10], Keystone [20],TIMBER-V [31] and CURE [2], which attempt to address the shortcomings of existing industry solutions, and steer the future for upcoming industry solutions.

In this paper, we provide an overview of recently proposed RISC-V enclave architectures, their advantages and limitations, and conclude with open challenges for future research in this exciting and foundational research area.

II. Background on Enclave Security Architectures

In the following section, we give an overview of enclave security architectures, their unifying characteristics and the

typically assumed adversary model against which they protect.
Overview. In Figure 1, a generic enclave security architecture is depicted. Enclave security architectures are deployed on systems which typically run a commodity OS that manages various applications. Most of the applications (Apps) are not severely security-sensitive and thus, the OS security mechanisms are considered enough to protect those applications. However, some applications (or services) on the system might be highly security-sensitive, e.g., because they process highly privacy-sensitive or IP-relevant data. The goal of an enclave architecture is to protect these sensitive services even in scenarios where the OS was compromised by an adversary.

On an abstract level, the enclave architecture achieves the protecting of sensitive services by providing isolated execution environments, called *enclaves*, as shown in Figure 1. The protection and isolation of the enclaves is backed by hardware-assisted security mechanisms which are implemented at the hardware level of the underlying platform. Depending on the specific enclave architecture, these mechanisms can be implemented at different locations, e.g., at the processor [10] or the system bus [2]. In all cases, the security mechanisms must be configured by a trusted software component (shown as Trusted SW in Figure 1) which typically represents the highest-privileged software running on the system. In some cases, e.g., Intel SGX [16] or AMD SEV [17], the trusted software component is implemented in microcode. By configuring the security mechanisms, the trusted software component effectively assigns system resources to enclaves, e.g., memory, processor cores or caches, depending on the specific capabilities of the enclave architecture.
Adversary Model. One of the key unifying characteristic of enclave security architectures is that they all assume a strong software adversary who is able to compromise the complete commodity software stack, including the OS kernel and even the hypervisor, if available. The adversary is assumed to spawn malicious processes and even malicious enclaves. Moreover, the adversary is capable of using system peripherals to perform Direct Memory Access (DMA) attacks [22]. In contrast to enclave architectures from industry [1], [16], [17], most academic solutions also consider the important attack class of cache side-channel attacks [6], [25], [26].

The adversary is not able to compromise the trusted software component which represents the Trusted Computing Base (TCB) of the system, together with the underlying hardware which is assumed to be correct and trusted. Denial-of-service attacks are typically not considered in enclave architectures since an adversary in control of the OS can trivially shut down the complete system. Physical attacks performed in close proximity to the system, e.g., fault injection attacks [3] or physical side-channel attacks [18], [21], are in general considered as out-of-scope for enclave architectures. However, some architectures, e.g., Intel SGX [16] or AMD SEV [17], provide a memory encryption engine which protects against simpler physical attacks such as cold-boot attacks [15] or attacks where an adversary reads out the content of the DRAM, e.g. by snooping the memory bus. An enclave architecture does not protect from software-exploitable vulnerabilities in the enclave code, instead, it prevents that their exploitation leads to a compromise of the complete system.

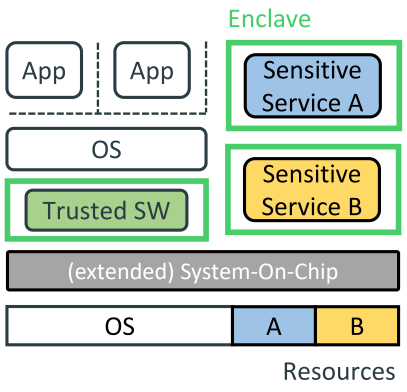

Fig. 1: Design overview of a generic enclave security architecture protecting the sensitive services A and B.

III. Enclave Security Architectures on RISC-V

We introduce next the most well-known academic RISC-V enclave architectures, namely, Sanctum [10], Keystone [20], TIMBER-V [31] and CURE [2]. We describe their design and features, and discuss their limitations. The assumed adversary models are aligned with the model described in Section II. We summarize our comparison in Table I.

A. *Sanctum*

In 2016, Costan et al. [10] proposed the Sanctum security architecture whose high-level design is shown in Figure 2.
Design, Features & Limitations. Sanctum offers *enclaves* to protect sensitive services on RISC-V platforms. In Sanctum, every enclave runs in the user level (as shown in Figure 2) and comes bundled with a non-sensitive application which invokes the enclave. The non-sensitive application can also just contain a code stub whose only purpose is invoking the enclave. Whenever an enclave is booted, the trusted software component of Sanctum, called the Security Monitor (SM), verifies the integrity of the enclaves using local attestation.

In Sanctum, the untrusted OS is utilized to manage the enclave memory and to provides typical OS services to the enclaves, e.g., interrupt handling or I/O services. In recent years, researchers showed that enclave architectures, which rely on untrusted software for enclave management tasks, are vulnerable to controlled side-channel attacks [23], [30], [34]. By observing the enclave page tables [34] or by interrupting enclave execution in a precise manner [30], an adversary can gather information about the internal state, e.g. control flow, of the enclave which can then be exploited to extract sensitive data, e.g. cryptographic key material, from the enclave. In Sanctum, similar attacks are prevented by storing the enclave page tables in enclave memory such that they cannot be directly observed by the OS. Moreover, enclaves are made aware of when they are interrupted and thus, can implement mechanisms to detect a suspicious interrupt behavior.

Sanctum implements two mechanisms to protect enclaves against cache side-channel attacks. Firstly, Sanctum flushes the L1 cache and Translation Lookaside Buffer (TLB) whenever a context switch into and out of an enclave is performed. Thus, an adversary cannot infer information about the internal enclave state by observing the state of the L1 cache or TLB. Secondly, the shared L2 cache is partitioned using a memory page coloring scheme which allows to assign cache lines exclusively to enclaves. However, the practicality of the scheme is limited since the assignment of cache lines to enclaves can only be set during system boot. A modification during runtime would require to completely rearrange the memory layout of the enclaves and even the OS.

Another limitation of Sanctum is that its enclaves can only execute unprivileged user-level code and thus, cannot contain device drivers. As a result, Sanctum cannot establish secure communication channels to peripherals which require unencrypted communication streams, e.g. sensors or GPUs. Sanctum provides a basic DMA attack protection by restricting DMA accesses, however, only a single system-wide DMA region can be defined for all DMA-capable devices.

Hardware Primitives & TCB. Sanctum's hardware-assisted security mechanisms are implemented at the Page Table Walker (PTW) which is part of the Memory Management Unit (MMU). The security mechanisms enforce that the OS cannot access any enclave memory and that an enclave cannot access the OS memory or memory regions of any other enclave by modifying its own page tables. The isolation is enforced by preventing a successful address translation of virtual memory addresses to physical addresses if the issuer of the memory request is not allowed to access the particular memory address. As typical for enclave architectures, all security critical operations are performed by the trusted software component (SM). In Sanctum, the SM runs in the machine level of the RISC-V processor. The basic DMA protection of Sanctum is implemented by adding two registers to the memory controller.

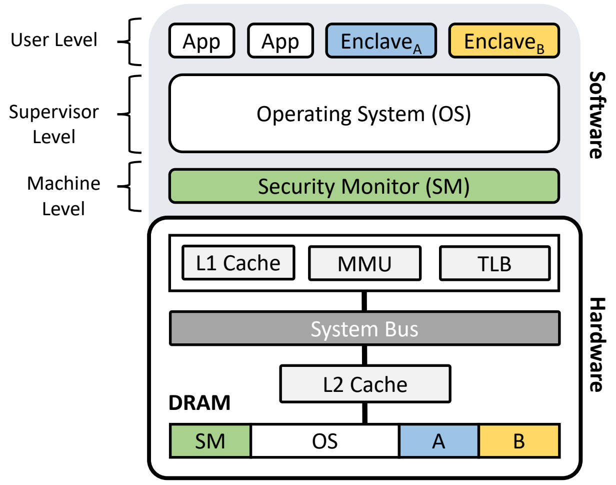

Fig. 2: Design overview of Sanctum which provides user-level enclaves and cache partitioning for the shared L2 cache through memory page coloring.

B. Keystone

The high-level design of the Keystone [20] security architecture is shown in Figure 3.

Design, Features & Limitations. In the Keystone security architecture, enclaves contain software from the user and supervisor privilege levels, as shown in Figure 3, whereby the enclave part running in the user-level is called the enclave app (EApp) and the part running in the supervisor level the enclave runtime. The enclave app contains the sensitive application code, whereas the enclave runtime provides typical OS services to the enclave app, e.g., memory management and interrupt handling. By including management tasks into the enclave, Keystone can more easily protect against the aforementioned controlled side-channel attacks which target page tables [34] or interrupt handlers [30].

In contrast to Sanctum, Keystone cannot schedule its enclaves like normal processes since they are not under the management of the OS. This means whenever an enclave is started, the OS state must be stored, the current processor core freed from the control of the OS and the enclave runtime booted which introduces an additional performance overhead. From a resource perspective, the enclave runtime (every enclave requires an own instance) leads to code duplication on the system and increases the overall memory consumption.

In theory, Keystone can include device drivers into the enclave runtime to establish secure unencrypted communication channels from enclaves to peripherals. However, for more complex peripherals, which communicate with the processor over DMA memory, additional mechanisms are required to control the DMA accesses to enclave memory, we call this functionality *enclave-to-peripheral binding*. Keystone does not support enclave-to-peripheral binding and cannot protect enclaves from DMA attacks [22].

When the executed enclaves (EApp + enclave runtime) are small in size, Keystone can protect them from simple hardware attacks, e.g. bus snooping, by executing them solely from a dedicated on-chip scratchpad memory. However, this only works for small applications since the size of a scratchpad memory is typically in the area of several hundreds of KB.

Hardware Primitives & TCB. In contrast to Sanctum, Keystone does not perform access control inside of the MMU but instead at a downstream hardware component called the Physical Memory Production (PMP) unit [13]. As detailed in the specification, the PMP can be used to define memory access permissions for the combination of user and supervisor level software and for the machine level software. Originally designed to protect the machine level software from the lesser privileged software, the Keystone authors utilize the PMP to assign a continuous memory region to each enclave and the trusted software component, called Security Monitor (SM). All memory regions not explicitly assigned automatically belong to the OS. The SM configures the PMP and represents the software TCB of the system.

Keystone prevents cache side-channel attacks on the enclaves by implementing a way-based partitioning scheme in the shared L2 cache which allows to assign cache ways exclusively to enclaves. However, since always complete cache ways are assigned, an underutilization of the cache resources might occur since the unused cache lines of an assigned way cannot be allocated by any other software component.

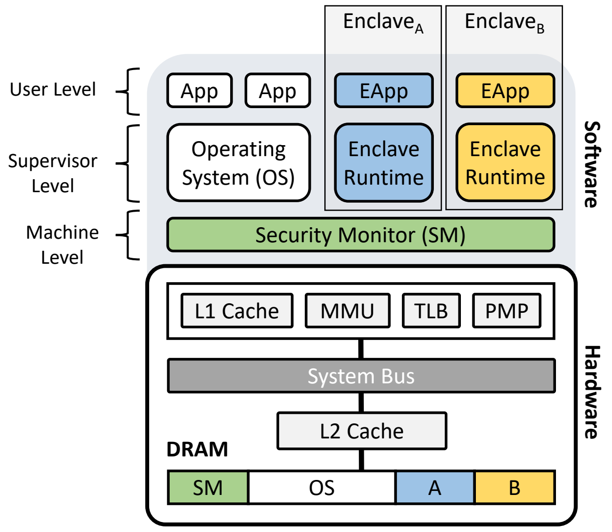

Fig. 3: Design overview of Keystone which provides enclaves that consist of an user-level enclave app (EApp) and a supervisor-level runtime.

C. TIMBER-V

The high-level design of TIMBER-V [31], which targets resource-constraint microcontrollers, is shown in Figure 4.

Design, Features & Limitations. TIMBER-V, in contrast to Sanctum and Keystone, does not isolate complete applications inside of enclaves. Instead, TIMBER-V only encapsulates parts of an application, e.g., a cryptographic library or a single cryptographic function, inside of its enclaves, called *trusted* domain. The rest of the application is called *normal* domain. TIMBER-V achieves the separation of the domains in memory using memory tagging which allows to define very fine-grained enclaves (we also call them *sub-level* enclaves) that might only comprise a few bytes of memory. The separation between the sub-level enclaves in user level is managed by a trusted domain called *TagRoot* which configures TIMBER-V's memory tagging hardware. Moreover, the TagRoot is used for setting up the enclaves (together with the OS) and provides services to the enclaves, e.g., sealing, attestation and a communication channel to the respective normal domains which is established over shared memory.

In theory, device drivers can be included into the TagRoot to allow enclaves to communicate with peripherals, e.g. sensors, over secure unencrypted channels. However, by adding more software to the TagRoot, the design of TIMBER-V diverges to the high-level security model of ARM's TrustZone-A [1] in which trusted applications are managed by a trusted OS. In recent years, academia has shown multiple times, summarized by Cerdeira et al. [7], that this trust model largely increases the attack surface of the system.

The handling of the enclave interrupts is not performed by the TagRoot but by the OS. Since the enclaves are not aware of their interruption, TIMBER-V is vulnerable to controlled side-channel attacks which target the interrupt handling [30].

Cache side-channel attacks are not considered by the authors of TIMBER-V since they argue that microcontrollers, which TIMBER-V targets, in general do not include cache memory.

Hardware Primitives & TCB. Memory tagging is implement in TIMBER-V inside the Memory Protection Unit (MPU) of the system. The MPU extension achieves not only that all applications are separated from each other but also that the normal and trusted domain parts of one applications are separated. Besides the MPU modifications, a hardware tag engine is required which verifies every memory access using the tags stored in memory. For every 32 bit of memory, 2 bit tags are required which leads to an memory overhead of 6.25%. Moreover, custom instructions are introduced which are needed for checking and manipulating the tags. If protection from DMA attacks [22] is required, additional tag engines must be placed at every DMA-capable peripheral.

The software TCB of the system is represented by the TagRoot, which verifies the MPU configuration set by the OS, and all software running in the machine level of the processor.

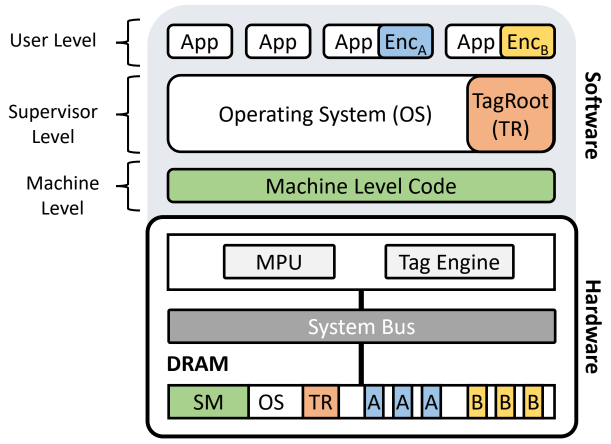

Fig. 4: Design overview of TIMBER-V which provides sub-level enclaves on resource-constraint microcontrollers.

D. CURE

The high-level design of the CURE [2] security architecture is shown in Figure 5.

Design, Features & Limitations. In contrast to the enclave architectures described before, CURE provides multiple *types* of enclaves from which an application developer who wants to protect his application can choose from. The different enclave types are shown in Figure 5, whereby Enclave$_A$ represents an user-level enclave, Enclave$_B$ an user/supervisor-level enclave and the Security Monitor (SM) a sub-level enclave. The main advantage of implementing the SM as a sub-level enclave is that the system TCB gets substantially reduced since all the security-irrelevant code in the machine level is excluded from the TCB. By providing different enclave types on one

Name	Enclave Type			Dynamic Cache Side-Channel Resilience	Controlled Side-Channel Resilience	Enclave-to-Peripheral Binding
	User-Level	User/Supervisor-Level	Sub-Level			
Sanctum [10]	●	○	○	◐	●	○
Keystone [20]	○	●	○	●	●	○
TIMBER-V [31]	○	○	●	○	◐	○
CURE [2]	●	●	●	●	●	●

TABLE I: Comparison of academic RISC-V enclave architectures.

platform, there is no need for a developer to adapt his sensitive application to the features and requirements of a specific enclave type. Instead, the developer can select the enclave type which best fits the needs of his sensitive application.

CURE protects user-level enclaves from controlled side-channel attacks targeting page tables [34] by including the enclave memory pages in the enclave memory and by verifying all page table modifications requested by the OS. Regarding side-channel attacks that target interrupt handlers [30], CURE allows enclave developers to define trap handlers which are automatically called after an enclave was interrupted and the interrupt handled by the OS. Thus, an enclave can use heuristics to detect an abnormal interrupt behavior.

CURE's user/supervisor-level enclaves can include device drivers into their runtime. Together with CURE's hardware security mechanisms this allows to assign peripherals exclusively to enclaves (enclave-to-peripheral binding).

CURE introduces a new cache architecture for shared caches which allows to assign cache ways exclusively to enclaves. In contrast to Sanctum's partitioning mechanism, the cache resources assigned to a particular enclave can be dynamically changed during runtime. However, as for the partitioning provided by Keystone, assigning complete cache ways to enclaves can lead to cache memory underutilization because of the coarse-grained nature of a cache way. Similar to Sanctum, CURE protects from cache side-channel attacks on the core-exclusive cache structures by flushing the L1 cache and TLB whenever an enclave is entered or exited.

Hardware Primitives & TCB. The key hardware security mechanism introduced with CURE is the filter engine which is included into the system bus. The filter engine allows, on the one hand, to assign memory regions exclusively to enclaves, and on the other hand, to define for every peripheral whether a certain enclave is allowed to communicate with it over MMIO. Moreover, the filter engine adds registers and control logic in front of all DMA-capable devices to restrict their access to parts of the memory. Thus, the filter engine enables a secure, yet unencrypted, communication between enclaves and (DMA-capable) devices over MMIO and shared DMA memory.

CURE's software TCB, the SM, is minimal in size since it only includes the security-relevant code and excludes the commodity firmware code typically also running in the machine level. The SM manages the enclaves and performs all security-critical operations, e.g., verification of the enclave binaries, key management or storing the enclave states persistently.

Besides the filter engine and the novel cache architecture, CURE requires minimal modifications at the RISC-V processor. In CURE, every enclave is identified by a system-wide unique ID. All access control operations done in CURE are performed based on the enclave ID. Thus, a new machine-level register is added to the processor to store the ID of the currently executed enclave. Moreover, the bus protocol (TileLink [28]) is extended by an enclave ID signal to propagate the ID through the complete system, from the processor over the cache controller up to the filter engine.

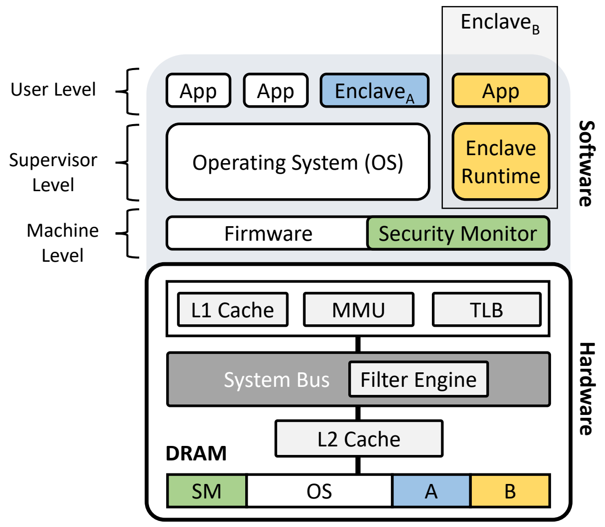

Fig. 5: Design overview of CURE which provides different *types* of enclaves, namely, sub-level enclaves (Security Monitor), user-level enclaves (Enclave$_A$) and user/supervisor-level enclaves (Enclave$_B$).

IV. Open Challenges

In the following, we discuss open challenges of enclave security architectures.

Side-channel Resilient Caches. As mentioned, none of the currently existing industrial enclave architectures [1], [16], [17] consider side-channel attacks in their threat model. In contrast, most academic proposals consider cache side-channel attacks. However, the proposed protection schemes are either impractical [10] or provide only a coarse-grained allocation of cache resources to enclaves [2], [20]. Thus, an open challenge in security research is to design practical and customizable cache microarchitectures [12], [27], [32] specifically for enclave architectures which allow to assign cache resources to enclaves in a fine-grained manner.

Enclaves on Emerging Platforms. The presented RISC-V security architectures mainly target platforms ranging from resource-constrained microcontrollers to more powerful embedded systems, thereby fitting into the main deployment area of RISC-V processors today. Regarding the resource-constrained microcontrollers, research should put a bigger focus on the energy-efficiency of those devices and how it is influenced by the hardware primitives introduced by security architectures. Moreover, if RISC-V expands its reach to large-scale cloud servers, novel enclave architectures need to be designed that can deal with heterogeneous computing platforms which might comprise hundreds of processor cores and a combination of CPUs, GPUs or Tensor Processing Units (TPUs). Designing enclave architectures that can protect sensitive applications also on platforms which require Network-on-Chip (NoC) bus architectures to connect a large number of computing nodes is another open research challenge.

Establish Trust in Hardware. For all enclave architectures introduced in this paper, the underlying hardware is inherently trusted. However, also hardware implementations can contain flaws which might diminish or completely dissolve the security guarantees of the system. Unfortunately, hardware, unlike software, cannot be patched once fabricated in silicon. This means that the security analysis required at the pre-silicon phase must be even more rigorous to verify the security of the hardware. As we have witnessed when conducting the largest international System-on-Chip (SoC) security competitions from 2018 up to now, called Hack@DAC [8] and Hack@SEC [9], hardware security analysis is very challenging to achieve in practice, even with state-of-the-art verification and analysis techniques. Thus, developing new methods and technologies to detect vulnerabilities in hardware implementations prior to fabrication is another important open research challenge.

References

[1] ARM Limited. Security technology: building a secure system using TrustZone technology. http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf, 2008.

[2] Raad Bahmani, Ferdinand Brasser, Ghada Dessouky, Patrick Jauernig, Matthias Klimmek, Ahmad-Reza Sadeghi, and Emmanuel Stapf. CURE: A Security Architecture with CUstomizable and Resilient Enclaves. In *30th USENIX Security Symposium (USENIX Security 21)*, 2021.

[3] Ingrid Biehl, Bernd Meyer, and Volker Müller. Differential fault attacks on elliptic curve cryptosystems. In *Annual International Cryptology Conference*, pages 131–146. Springer, 2000.

[4] Ferdinand Brasser, Brahim El Mahjoub, Ahmad-Reza Sadeghi, Christian Wachsmann, and Patrick Koeberl. Tytan: tiny trust anchor for tiny devices. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2015.

[5] Ferdinand Brasser, David Gens, Patrick Jauernig, Ahmad-Reza Sadeghi, and Emmanuel Stapf. Sanctuary: Arming trustzone with user-space enclaves. In *NDSS*, 2019.

[6] Ferdinand Brasser, Urs Müller, Alexandra Dmitrienko, Kari Kostiainen, Srdjan Capkun, and Ahmad-Reza Sadeghi. Software grand exposure: Sgx cache attacks are practical. In *11th USENIX Workshop on Offensive Technologies (WOOT 17)*, 2017.

[7] David Cerdeira, Nuno Santos, Pedro Fonseca, and Sandro Pinto. Sok: Understanding the prevailing security vulnerabilities in trustzone-assisted tee systems. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P), San Francisco, CA, USA*, pages 18–20, 2020.

[8] Design Automation Conference. Hack@DAC 2020. https://hackat.events/dac20/, 2020.

[9] USENIX Security Conference. Hack@SEC 2020. https://hackat.events/sec20/, 2020.

[10] Victor Costan, Ilia A Lebedev, and Srinivas Devadas. Sanctum: Minimal Hardware Extensions for Strong Software Isolation. In *USENIX Security Symposium*, 2016.

[11] Lucas Davi et al. HAFIX: Hardware-Assisted Flow Integrity eXtension. In *ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2015.

[12] Ghada Dessouky, Tommaso Frassetto, and Ahmad-Reza Sadeghi. HybCache: Hybrid Side-Channel-Resilient Caches for Trusted Execution Environments. In *USENIX Security 20*, 2020.

[13] RISC-V Foundation. The risc-v instruction set manual, volume ii: Privileged architecture, documentversion 20190608-priv-msu-ratified”. https://riscv.org/specifications/privileged-isa/, 2019.

[14] Tommaso Frassetto, Patrick Jauernig, Christopher Liebchen, and Ahmad-Reza Sadeghi. Imix: In-process memory isolation extension. In *USENIX Security 2018*, pages 83–97, 2018.

[15] J Alex Halderman, Seth D Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A Calandrino, Ariel J Feldman, Jacob Appelbaum, and Edward W Felten. Lest we remember: cold-boot attacks on encryption keys. *Communications of the ACM*, 52(5):91–98, 2009.

[16] Intel. Intel Software Guard Extensions Programming Reference. https://software.intel.com/sites/default/files/managed/48/88/329298-002.pdf, 2014.

[17] David Kaplan, Jeremy Powell, and Tom Woller. Amd memory encryption. https://developer.amd.com/wordpress/media/2013/12/AMD_Memory_Encryption_Whitepaper_v7-Public.pdf, 2016.

[18] Paul C Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *Annual International Cryptology Conference*, pages 104–113. Springer, 1996.

[19] Patrick Koeberl, Steffen Schulz, Ahmad-Reza Sadeghi, and Vijay Varadharajan. Trustlite: A security architecture for tiny embedded devices. In *Proceedings of the Ninth European Conference on Computer Systems*, page 10. ACM, 2014.

[20] Dayeol Lee, David Kohlbrenner, Shweta Shinde, Krste Asanović, and Dawn Song. Keystone: An open framework for architecting trusted execution environments. In *Proceedings of the Fifteenth European Conference on Computer Systems*, pages 1–16, 2020.

[21] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks: Revealing the secrets of smart cards*, volume 31. Springer Science & Business Media, 2008.

[22] A Theodore Markettos, Colin Rothwell, Brett F Gutstein, Allison Pearce, Peter G Neumann, Simon W Moore, and Robert NM Watson. Thunderclap: Exploring vulnerabilities in operating system iommu protection via dma from untrustworthy peripherals. In *NDSS*, 2019.

[23] Mathias Morbitzer, Manuel Huber, Julian Horsch, and Sascha Wessel. Severed: Subverting amd’s virtual machine encryption. In *Proceedings of the 11th European Workshop on Systems Security*. ACM, 2018.

[24] Job Noorman, Pieter Agten, Wilfried Daniels, Raoul Strackx, Anthony Van Herrewege, Christophe Huygens, Bart Preneel, Ingrid Verbauwhede, and Frank Piessens. Sancus: Low-cost trustworthy extensible networked devices with a zero-software trusted computing base. In *22nd USENIX Security symposium*, 2013.

[25] Dag Arne Osvik, Adi Shamir, and Eran Tromer. Cache attacks and countermeasures: the case of AES. In *RSA Conference*, 2006.

[26] Colin Percival. Cache missing for fun and profit, 2005.

[27] Moinuddin K. Qureshi. Ceaser: Mitigating Conflict-based Cache Attacks via Encrypted-Address and Remapping. In *MICRO 18*, 2018.

[28] SiFive. Sifive tilelink specification. https://sifive.cdn.prismic.io/sifive%2F57f93ecf-2c42-46f7-9818-bcdd7d39400a_tilelink-spec-1.7.1.pdf, 2018.

[29] Trusted Computing Group. Tpm 1.2 protection profile. https://www.trustedcomputinggroup.org/tpm-1-2-protection-profile/, 2016.

[30] Jo Van Bulck, Frank Piessens, and Raoul Strackx. Nemesis: Studying microarchitectural timing leaks in rudimentary cpu interrupt logic. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 178–195. ACM, 2018.

[31] Samuel Weiser, Mario Werner, Ferdinand Brasser, Maja Malenko, Stefan Mangard, and Ahmad-Reza Sadeghi. Timber-v: Tag-isolated memory bringing fine-grained enclaves to risc-v. In *NDSS*, 2019.

[32] Mario Werner, Thomas Unterluggauer, Lukas Giner, Michael Schwarz, Daniel Gruss, and Stefan Mangard. ScatterCache: Thwarting Cache Attacks via Cache Set Randomization. In *USENIX Security 19*, 2019.

[33] Jonathan Woodruff, Robert NM Watson, David Chisnall, Simon W Moore, Jonathan Anderson, Brooks Davis, Ben Laurie, Peter G Neumann, Robert Norton, and Michael Roe. The cheri capability model: Revisiting risc in an age of risk. In *ISCA 2014*. IEEE, 2014.

[34] Yuanzhong Xu, Weidong Cui, and Marcus Peinado. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In *2015 IEEE S&P*, pages 640–656. IEEE, 2015.